[redacted]

I l 6r
no. 992
cop. 2

UIUCDCS-R-79-992

UILU-ENG 79 1740

Quadratic Algorithms for Minimizing Joins in Restricted Relational Expressions

by

Yehoshua Sagiv

October 1979

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-79-992

Quadratic Algorithms for Minimizing Joins in Restricted Relational Expressions

Yehoshua Sagiv

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

October 1979

This work was supported in part by the National Science Foundation under grant MCS-76-15255

ABSTRACT

An important step in the optimization of queries in relational databases is minimizing the number of join operations used in the evaluation of a query. It is shown that three subclasses of tableaux (including the subclass of simple tableaux) have $O(n^2)$ time equivalence and minimization algorithms. Since tableaux are nonprocedural representations of relational expressions over select, project and join, these minimzation algorithms can be used to minimize the number of join operators in expressions whose tableaux belong to one of these subclasses.

CR categories: 4.33, 5.25

Key words and phrases: relational database, relational algebra, query optimization, equivalence of queries, conjunctive query, tableau, NP-complete.

1. Introduction

The relational model for databases features two high-level query languages: the relational algebra and the relational calculus [9,10]. The relational algebra is a procedural language that uses operators defined on relations, and hence a query is usually translated to a relational expression before being evaluated. However, the efficiency with which a query can be answered depends on the relational expression that has been chosen to represent this query. Consequently, a number of papers (e.g., [12,13,14,15,17,18]) have considered transformations that reduce the cost of evaluating a query. However, these transformations do not necessarily produce an equivalent query of least cost. Chandra and Merlin [8] show how to perform global optimization on a large class of queries, but their algorithm is exponential in the size of the query.

The most commonly used operators of the relational algebra are select, project and join, and a polynomial time algorithm for optimizing a subclass of expressions with these operators is given in [4,5]. This optimization technique uses tableaux [3] as a nonprocedural representation of queries. Tableaux are similar to the conjunctive queries of [8], and resemble Zloof's "Query-by-Example" language [20]. Relational expressions over select, project, and join can be represented by tableaux [3]. In [8] it is shown that every tableau has an equivalent minimal tableau. The importance of this result follows from the fact that an expression with a minimum number of joins corresponds to a minimal tableau. A polynomial minimization algorithm for the class of simple tableau is given in [4] (although the problem is NP-complete in the general case [3]). In [5] it shown how to obtain in polynomial time an optimal expression from a minimal simple tableau (if such an expression exists).

This optimization technique is machine independent. It is capable of minimzing the number of join operators (note that join is the most expensive operator to implement), eliminating redundant subexpressions, and applying select and project as early as possible. In a relational database system, this type of optimization should be augmented with machine dependent optimization that takes into consideration the size of the relations involved, sorted columns, etc.

In this paper we describe new minimization algorithms for two subclasses of tableaux. We also show how to improve the running time of the minimzation algorithm for the class of simple tableaux. All these algorithms have an $O(n^2)$ running time. It shown that each one of the three subclasses of tableaux discussed in this paper also has an $O(n^2)$ time equivalence algorithm. Finally, in Sections 7 and 8 we touch upon the problems of minimizing tableaux in the general case, and obtaining optimal expressions from minimal tableaux.

2. Basic Definitions

2.1 The Relational Model

The relational model for databases [9] assumes that the data is stored in tables called <u>relations</u>. The columns of a table correspond to attributes, and the rows to records or tuples. Each attribute has an associated domain of values. A tuple is viewed as a mapping from the attributes to their domains, since no canonical ordering of the attributes is needed in this way. If r is a relation with a column corresponding to the attribute A, and μ is a tuple in r, then $\mu(A)$ is the value of the A-component of μ. In this paper we usually denote a relation as a set of tuples.

A relation scheme is a set of attributes labeling the columns of a table, and it is usually written as a string of attributes. We often use the relation scheme itself as the name of the table. A relation is just the "current value" of a relation scheme. The relation is said to be defined on the set of attributes of the relation scheme.

2.2 The Relational Algebra and Relational Expressions

The relational algebra [9,10] is a set of operators defined on relations. In this paper we consider the operators select, project, and join.

Let r be a relation defined on a set of attributes X, A an attribute in X and c a value from the domain of A. The selection A=c, written $\sigma_{A=c}(r)$, is

$$\sigma_{A=c}(r) = \{\mu \mid \mu \text{ is a tuple in r and } \mu(A)=c\}$$

Let Y be a subset of X, the projection of r onto Y, written $\pi_Y(r)$, is

$$\pi_Y(r) = \{\mu \mid \mu \text{ is a mapping on } Y, \text{ and there is a } \nu \text{ in } r \text{ that agrees with } \mu \text{ on } Y\}$$

Let r_1 and r_2 be relations defined on the relation schemes R_1 and R_2, respectively. The (natural) join of r_1 and r_2, written $r_1 \bowtie r_2$, is

$$r_1 \bowtie r_2 = \{\mu \mid \mu \text{ is a mapping on } R_1 \cup R_2, \text{ and there is } \nu_1 \text{ in } r_1 \text{ that agrees with } \mu \text{ on } R_1, \text{ and } \nu_2 \text{ in } r_2 \text{ that agrees with } \mu \text{ on } R_2\}$$

Note that the join includes intersection (when R_1 and R_2 are the same) and cartesian product (when R_1 and R_2 are disjoint) as special cases.

The relational algebra includes other operators, and to make our set of operators "complete" (in the sense of Codd [10]) we only need to add the union and difference operators, and allow selections involving arithmetic comparisons between two components of a tuple.

The operators of the relational algebra are used in the formulation of queries in terms of relational expressions. A restricted relational expression consists of select, project, and join as operators, and relation schemes as operands.

2.3 Expression Values and Equivalence of Expressions

An underlying assumption in many papers (e.g., [1,6,7]) is the existance of a single universal relation at each instant of time. This relation is defined on the set of all the attributes, and it is called a universal instance or just an instance. If I is an instance and E is a relational expression, then each relation scheme R_i in E is assigned the relation $\pi_{R_i}(I)$. The value of E with respect to I, written $v_I(E)$, is computed by applying operators to operands in the following natural way.

(1) If E is a single relation scheme R_i, then $v_I(E) = \pi_{R_i}(I)$.

(2) (a) If $E = \sigma_{A=c}(E_1)$, then $v_I(E) = \sigma_{A=c}(v_I(E_1))$.

(b) If $E = \pi_X(E_1)$, then $v_I(E) = \pi_X(v_I(E_1))$.

(c) If $E = E_1 \bowtie E_2$, then $v_I(E) = v_I(E_1) \bowtie v_I(E_2)$.

We may regard a relational expression as a mapping from instances to relations, i.e., the expression E maps the instance I to the relation $v_I(E)$. Two expressions E_1 and E_2 are <u>equivalent</u> if they define the same mapping. That is, if for all instances I, $v_I(E_1) = v_I(E_2)$. In [3] other types of equivalence are discussed, and it is shown that results obtained for the above type of equivalence also apply to the more general case, where the relations assigned to the relation schemes do not necessarily come from an instance.

3. Tableaux

Tableaux are just another way of representing mappings from instances to relations. Unlike relational expressions, tableaux are nonprocedural representation of queries in exactly the sense that relational calculus [9,10] is nonprocedural. In [3] it is shown that every restricted relational expression has a corresponding tableau that represents the same mapping.

A tableau is a matrix consisting of a summary and a set of rows. The columns of a tableau correspond to the attributes of the universe in a fixed order. The symbols appearing in a tableaux are chosen from

(1) Distinguished variables, usually denoted by subscripted a's.

(2) Nondistinguished variables, usually denoted by subscripted b's.

(3) Constats, which are drawn from the domains of the attributes.

(4) Blank.

Each row may contain constants, distinguished and nondistinguished variables. The summary has constants, distinguished variables and blanks. A variable cannot appear in more than one column, and a distinguished variable may appear in a particular column only if it appears in the summary. Furthermore, if a constant or a distinguished variable appears in some column A of the summary, then it must also appear in column A of at least one row.

Let T be a tableau with a summary w_0 and rows $w_1,\ldots,w_n$, and let S be the set of all the nonblank symbols in T. A valuation ρ for T maps each symbol in S to a constant, such that if c is a constant in S, then $\rho(c)=c$. The valuation ρ is extended to the rows and summary of T by defining $\rho(w_i)$ to be the result of substituting $\rho(v)$ for all variables v in w_i.

A tableau T defines a mapping from instances to relations on its target relation scheme, which is the set of all the attributes corresponding to columns that have a nonblank symbol in the summary. Given an instance I, the value of T, written T(I), is

$T(I) = \{\rho(w_0) \mid \text{for some valuation } \rho, \text{ we have } \rho(w_i) \text{ in } I \text{ for } 1<i<n\}$

Conventionally, we also regard ϕ as a tableau. This tableau represents the function that maps every instance to the empty relation.

Example 1: Consider the following tableau.

$$T = \begin{array}{|ccc|} \multicolumn{1}{c}{A} & \multicolumn{1}{c}{B} & \multicolumn{1}{c}{C} \\ \hline a_1 & & a_3 \\ \hline a_1 & 2 & b_3 \\ b_1 & b_2 & a_3 \\ a_1 & b_2 & b_4 \\ \hline \end{array}$$

The summary is shown first, with a line below it, and integers are used as constants.

Tableau T defines a relation on the relation scheme AC. For example, supoose that I is the instance {211,121,122}.

Consider the valuation ρ which assigns 2 to b_2 and 1 to every other variable in T. Under this valuation, each row of T becomes 121, which is a member of I. Therefore, $\rho(a_1a_3) = 11$ is in T(I).

If ρ assigns 1 to a_1, b_1, b_3 and b_4, and 2 to b_2 and a_3, the first and third rows become 121 and the second row becomes 122; both are members of I, and so 12 is in T(I).

Since no valuation for T produces a tuple other than 11 or 12, T(I) = {11,12}. []

3.1 Relational Expressions and Tableaux

Every restricted relational expression E has a corresponding tableau T that defines the same mapping as E (however, the converse is not true) [3]. The tableau T for the expression E can be constructed bottom up by applying the following rules [3].

(A) If there are no operators in E, then E is a single relation scheme R. The tableau T for E has one row and a summary such that:

(i) If A is an attribute in R, then in the column for A, the tableau T has the same distinguished variable in the summary and row.

(ii) If A is not in R, then its column has a blank in the summary and a nondistinguished variable in the row.

(B1) Suppose E is of the form $\sigma_{A=c}(E_1)$, and we have constructed T_1, the tableau for E_1.

(i) If the summary of T_1 has blank in the column for A, then the expression E has no meaning, and the tableau T for E is undefined.

(ii) If T_1 has a constant $c' \neq c$ in the summary column for A, then for any instance I, $v_I(E)$ has only tuples with c' in the component for A, and $v_I(E)$ is ϕ. Hence, the tableau for E is ϕ.

(iii) If T_1 has c in the summary column for A, then the tableau for E is T_1.

(iv) If T_1 has a distinguished variable a in the summary column for A, the tableau T for E is constructed by replacing a by c whenever it appears in T_1.

(B2) Suppose E is of the form $\pi_X(E_1)$ and T_1 is the tableau for E_1. Construct T for E by replacing nonblank symbols by blanks in the summary of T_1 for those columns whose attributes are not in X. Distinguished variables in those columns become nondistinguished.

(B3) Suppose E is of the form $E_1 \bowtie E_2$, and T_1 and T_2 are the tableaux for E_1 and E_2, respectively.

(i) If T_1 and T_2 have some column in which their summaries have distinct constants, then it easy to show that for all instances I, $v_I(E)= \phi$, so ϕ is the tableau for E.

(ii) If no corresponding positions in the summaries have distinct constants, let S_1 and S_2 be the sets of symbols of T_1 and T_2, respectively. We may take S_1 and S_2 to have disjoint sets of nondistinguished variables, but identical distinguished variables in corresponding columns. Construct T for E to have the union of all the rows of T_1 and T_2. The summary of T has in a given column:

(a) The constant c if one or both T_1 and T_2 have c in that column's summary. In this case replace any distinguished variable in that column by c.

(b) The distinguished symbol a if (a) does not apply, but one or both of T_1 and T_2 have a in that column's summary.

(c) Blank, otherwise.

These rules can also be used to define the operations select, project and join on tableaux. The result of applying any one of these operators to tableaux (not necessarily tableaux derived from restricted relational expressions) is defined to be the tableau described in the rule for that operator.

Example 2: Consider the expression $\pi_{AC}(\pi_{AC}(AB \bowtie BC) \bowtie \sigma_{B=2}(AB))$ If the above rules are applied to this expression, then the result is the tableau of Example 1. []

3.2 Equivalence of Tableaux

Two tableaux T_1 and T_2 are equivalent, written $T_1 \equiv T_2$, if for all instances I, $T_1(I) = T_2(I)$. We say that T_1 is contained in T_2, written $T_1 \subseteq_T T_2$, if for all I, $T_1(I) \subseteq T_2(I)$.

Let T_1 and T_2 be tableaux with the same target relation scheme, and let S_1 and S_2 be the sets of symbols of T_1 and T_2, respectively. A homomorphism is a mapping $\xi: S_1 \to S_2$ such that:

(a) If c is a constant, then $\xi(c)=c$.

(b) If s is the summary of T_1, then $\xi(s)$ is the summary of T_2.

(c) If w is any row of T_1, then $\xi(w)$ is a row of T_2.

The following theorem is proved in [3,8].

Theorem 1: $T_2 \subseteq_T T_1$ if and only if T_1 and T_2 have the same target relation scheme, and there is a homomorphism $\xi: S_1 \to S_2$.

By condition (c), a homomorphism ξ corresponds to a mapping θ from the rows of T_1 to the rows of T_2. The mapping θ is called a containment mapping, and it satisfies the following conditions.(1)

---

(1) If row w of T_1 has a constant in some column A, then $\theta(w)$ has the same constant in column A.

(2) If row w of T_1 has a distinguished variable in column A, then $\theta(w)$ has a distinguished variable in column A.

(3) If rows w and v have the same nondistinguished variable in column A, then rows $\theta(w)$ and $\theta(v)$ have the same symbol in column A.

Corollary 2: [3] Tableaux T_1 and T_2 are equivalent if and only if they have identical summaries up to renaming of distinguished variables, and there are containment mappings in both directions.

A tableau T is minimal if T is not equivalent to any tableau with fewer rows. Note that if T comes from an expression E, then the number of joins in E is one less than the number of rows in T. Thus, if T is minimal, it corresponds to an expression with a minimum number of joins. For every tableau T, there is a unique (up to renaming of variables) tableau T′, such that T ≡ T′ and T′ is minimal [8]. Furthermore, the minimal tableau equivalent to T can be obtained by deleting some of the rows of T. The core(2) of T is the set of all the rows that belong to the minimal tableau obtained by deleting redundant rows of T. A folding is a containment mapping from the rows of a tableau T to the rows of the core of T, such that every row in the core of T is mapped to itself. It can be shown that every tableau T has a folding [8]. Note that a

(1) In this paper we consider only equivalence (and not proper containment) of tableaux, and therefore the definition of a containment mapping is more restricted than the original definition given in [3].

(2) A tableau T might have two different cores. However, they are the same up to renaming of variabls. Therefore, we usually speak about the core of T.

homomorphism that corresponds to a folding maps every variable in the core of T to itself. The following corollary is an immediate consequence of the results stated so far.

Corollary 3: Let T_1 and T_2 be tableaux with the same summary. If the rows of T_1 are a subset of the rows of T_2, then

(1) $T_2 \subseteq_T T_1$, and

(2) $T_2 \equiv T_1$ if and only if the core of T_2 is contained in T_1.

Let w and x be rows of tableaux over the same set of attributes. Row w covers row x, written $x \leqslant w$, if for all columns A,

(1) if x has a constant in column A, then w has the same constant in column A, and

(2) if x has a distinguished variable in column A, then w has a distinguished variable in column A.

Note that if x is mapped to w, and $x \leqslant w$, then the first two conditions of a containment mapping are satisfied. Let R and S be sets of rows over the same set of attributes. The set S covers the set R, written $R \leqslant S$, if every row of R is covered by some row of S.

A symbol (i.e., a variable or a constant) of a tableau T is repeated in some column A, if it appears in that column in more than one row. A tableau T is simple if whenever T has a repeated nodistinguished variable b in some column A, then b is the only repeated symbol in that column. The class of simple tableaux has an $O(n^3)$ equivalence algorithm [3], and an $O(n^4)$ minimzation algorithm [4]. Other equivalence algorithms can be obtained from the containment algorithms of [16]. That is, tesing whether $T_2 \equiv T_1$ can be done in $O(n^2)$ in the following two cases:

(1) Both T_1 and T_2 have at most one repeated nondistinguished variable in each row.

(2) Every row of T_1 is covered by at most two rows of T_2, and every row of T_2 is covered by at most two rows of T_1.

4. Polynomial Equivalence Algorithms

In this section we consider the following classes of tableaux.

(1) The class of all tableaux T, such that T has at most one repeated nondistinguished variable in each row.

(2) The class of all tableaux T, such that every row of T is covered by at most one row besides itself.

Note that deciding whether a tableau belongs to Class 1 (or whether a tableau is simple) requires $O(n)$ time. However, deciding whether a tableau belongs to Class 2 requires $O(n^2)$.

Theorem 4: Each one of the above classes has an $O(n^2)$ time equivalence algorithm.

Proof: It follows immediately from the results of [16] that the theorem is true for Class (1). Suppose that both T_1 and T_2 are tableaux that belong to Class 2. We say that two rows w and x are equivalent, if w covers x and x covers w. Consider the algorithm of Figure 1. This algorithm tests whether $T_1 \equiv T_2$. Obviously, if $T_1 \equiv T_1'$ and $T_2 \equiv T_2'$,

```
begin
(1)     let $T_1'$ be the result of removing from $T_1$ every row
        that is not equivalent to some row of $T_2$;
(2)     let $T_2'$ be the result of removing from $T_2$ every row
        that is not equivalent to some row of $T_1$;
(3)     if $T_1 \not\equiv T_1'$ then return false;
(4)     if $T_2 \not\equiv T_2'$ then return false;
(5)     if $T_1' \equiv T_2'$ then return true else return false;
end
```

Figure 1

then $T_1 \equiv T_2$ if and only if $T_1' \equiv T_2'$. Thus, we have to show that T_1 and T_2 cannot be equivalent if either $T_1 \not\equiv T_1'$ or $T_2 \not\equiv T_2'$.

Suppose that $T_1 \not\equiv T_1'$. Since the rows of T_1' are a subset of the rows of T_1, it follows that the core of T_1 contains a row w which is not in T_1'. By the construction of T_1', no row of T_2 is equivalent to w. But equivalent tableaux have cores which are the same (up to renaming of variables). That is, every row in the core of one tableau has an equivalent row in the core of the other tableau. Therefore, T_1 and T_2 cannot be equivalent. Similarly, if $T_2 \not\equiv T_2'$, then $T_1 \not\equiv T_2$.

Obviously, for $i \in \{1,2\}$, each row of T_i is covered by at most two rows of T_i', and each row of T_i' is covered by at most two rows of T_i. Suppose that a row w of T_1' is covered by more than two rows of T_2'. Let x be a row of T_2' that is equivalent to w. Row x is covered by every row of T_2' that covers w. Therefore, x is covered by at least two rows of T_2' besides itself. This contradiction implies that each row of T_1' is covered by at most two rows of T_2'. Similarly, each row of T_2' is covered by at most two rows of T_1'.

It follows that testing equivalence in lines (3)-(5) can be done using the algorithm of [16]. Each application of this algorithm requires $O(n^2)$ time. Lines (1) and (2) can be executed in $O(n^2)$ time and, therefore, the algorithm of Figure 1 has a time complexity of $O(n^2)$. []

5. Obtaining Minimization Algorithms from Equivalence Algorithms

Let S be a class of tableaux. We say that S is _closed under row deletion_ if whenever a tableau T is in S, and T′ is obtained by deleting some of the rows of T, then T′ is also in S.

Theorem 5: Let S be a class of tableaux closed under row deletion. If there is an equivalence algorithm for S that runs in F(n) time ($F(n) \geq cn$ for some constant c), then there is a minimization algorithm for S that runs in nF(n) time.

Proof: Figure 2 describes a minimization algorithm for S. The input for this algorithm is a tableau T of S. We assume that the numbers 1,2,...,r correspond to the rows of T. The algorithm is based on the ability to test equivalence of tableaux from S. Note that the equivalence test in line (3) can be replaced with the containment test

$T' \underline{C}_T T$, because the rows of T' are a subset of the rows of T and, hence, $T \underline{C}_T T'$.

Let T_i be the value of T after i iterations through the loop of lines (1)-(3). Since T is assigned a new value in line (3) only if the new value is equivalent to the old value, T_r (i.e., the tableau returned by the algorithm) is equivalent to T_0. Suppose that T_r is not minimal. Therefore, there is a row j in T_r that does not belong to the core of T_r. Let $\overline{T}_r$ be the result of deleting row j from T_r. Since the core of T_r is not changed as a result of this deletion, $T_r \equiv \overline{T}_r$ and, hence, $\overline{T}_r \equiv T_0$.

Since row j has not been deleted by the algorithm, it must be in T_j. Let $\overline{T}_j$ be the result of removing row j from T_j. It follows that $\overline{T}_j$ is not equivalent to T_0 (otherwise row j cannot appear in T_r). But the rows of $\overline{T}_r$ are a subset of the rows of $\overline{T}_j$, and the rows of $\overline{T}_j$ are a subset of the rows of T_0 and, therefore, $T_0 \underline{C}_T \overline{T}_j \underline{C}_T \overline{T}_r$. Since $\overline{T}_r \equiv T_0$, it follows that $\overline{T}_j \equiv T_0$. This contradiction implies that T_r is minimal.

In each pass through the main loop, line (2) requires $O(n)$ time and line (3) requires $F(n)$ time. The loop is repeated r times to give a time complexity of $nF(n)$ (since $r \leq n$). []

6. Polynomial Minimization Algorithms

The classes of tableaux described in Section 4, and the class of simple tableux have polynomial time equivalence algorithms. Each one of

```
    begin
(1)     for i := 1 to r do
            begin
(2)             Let T′ be the result of deleting row i from T;
(3)             if T ≡ T′ then T := T′;
            end;
(4)     return T;
    end
```

Figure 2

these classes is closed under row deletion and, therefore, Theorem 5 can be applied to obtain polynomial minimization algorithms for these classes. However, the algorithms obtained by applying Theorem 5 are not the most efficient minimization algorithms for these classes. For each one of these classes there is a minimization algorithm with a time complexity $O(n^2)$. These minimization algorithms are given in the following sections.

6.1 Regular Tableaux

In this section we will show that a tableau, in which a folding does not eliminate any repeated nondistinguished variable, can be minimized in $O(n^2)$ time. This fact is used in developing the minimization algorithms of the following sections.

Let b a repeated nondistinguished variable of a tableau T. The variabel b is *essential* if it appears in every core of T. A tableau T is *regular* if all repeated nondistinguished variables of T are essential.

```
    begin
    /* The input is a tableau T. */
    /* Initially all the rows of T are marked "unconsidered". */
(1)     while there is a row w marked "unconsidered" do
            begin
(2)             mark w "considered";
(3)             for every row x other than w do
(4)                 if in whatever column x and w disagree,
                       x has a nondistinguished variable that appears
                       nowhere else in T then delete x from T;
            end;
(5)     return T;
    end
```

Figure 3

Lemma 6: The algorithm of Figure 3 returns a tableau $\overline{T}$ equivalent to T in $O(n^2)$ time. Furthermore, if T is regular, then $\overline{T}$ is minimal.

Proof: Consider line (4) of the algorithm. Row x is deleted if there is a row w in T, such that for all columns A, if x and w disagree in column A, then x has a nondistinguished variable that appears nowhere else in T. By Corollary 2 in [3], the tableau obtained by deleting row x from T is equivalent to T. Thus, $\overline{T}$ (the tableau returned by the algorithm) is equivalent to T. As for the running time of this algorithm, let T have r rows and t columns. The cost of executing line (4) once is O(t). In each iteration of the outer loop, the inner loop is executed at most r times. The outer loop is executed no more than r times. Thus, the total cost of line (4) is $O(r^2t)$. Every other line has a constant cost, and is executed no more than r^2 times. Since both rt and r are smaller than n, the algorithm has a $O(n^2)$ running time.

Suppose that T is regular. Every repeated nondistinguished variable of T is essential and, therefore, must appear in the core of T.

Consider a folding from T onto its core. This folding maps every row in the core of T to itself and, therefore, the corresponding homomorphism maps every repeated nondistinguished variable of T to itself. Suppose that row x of T is mapped to some other row w in the core of T. Since each repeated nondistinguished variable is mapped to itself, it follows that in whatever column x and w disagree, x has a nondistinguished variable that appears nowhere else in T. Therefore, x is deleted during the execution of the above algorithm. Since this is true for every x which is not in the core of T, $\overline{T}$ is minimal. []

Each one of the following algorithms for minimizing a tableau T has two steps. In the first step some rows of T are deleted in order to obtain an equivalent regular tableau $\overline{T}$. In the second step the algorithm for minimizing regular tableaux is applied to $\overline{T}$.

6.2 Minimizing Tableaux of Class 1

Let T be a tableau that has at most one repeated nondistinguished variable in each row. For each repeated nondistinguished variable b, let W(b) be the set of all the rows that contain b. Suppose that some repeated nondistinguished variable b_0 is not essential, and let A be the column in which b_0 appears. Let θ be a folding from the rows of T to its core. Obviously, all the rows of $\theta(W(b_0))$ have the same symbol d $(d \neq b)$ in column A, and $W(b_0)$ is covered by $\theta(W(b_0))$. The following lemma shows that these conditions are also sufficient for the elimination of b_0.

```
    begin
    /* The input is a tableau T that belongs to Class 1.             */
    /* Initially all the variables of T are marked "unconsidered". */
(1)     while there is a repeated nondistinguished
              variable b marked "unconsidered" do
          begin
(2)         mark b "considered";
(3)         let A be the column in which b appears;
(4)         for each symbol s (s ≠ b) in column A do
              begin
(5)             let S be the set of all the rows of T
                that have the symbol s in column A;
(6)             if W(b) ≤ S then delete W(b) from T;
              end;
          end;
(7)     return T;
    end
```

Figure 4

Lemma 7: Suppose that ψ is a mapping from T to itself such that for all $x \notin W(b_0)$, $\psi(x) = x$. Then ψ is a containment mapping if and only if (1) all the rows of $\psi(W(b_0))$ have the same symbol in column A, and (2) for all $x \in W(b_0)$, x is covered by $\psi(x)$.

Proof: Obviously, conditions (1) and (2) are satisfied if ψ is a containment mapping. Conversely, if ψ satisfies condition (2), then ψ satisfies the first two conditions of a containment mapping. If two rows x and w of T have the same nondistinguished variable b in some column B, then either both x and w are in $W(b_0)$ and b is b_0, or both x and w are not in $W(b_0)$. In either case, $\psi(x)$ and $\psi(w)$ have the same symbol in column B. Thus, ψ is a containment mapping. []

By Lemma 7, if b_0 is not essential, then we can always delete all the rows containing b_0 by finding a set of rows S, such that S covers $W(b_0)$ and all the rows of S have the same symbol in column A. The algorithm of Figure 4 computes a regular tableau T′ equivalent to T in this way.

Theorem 8: A tableau T that has at most one repeated nondistinguished variable in each row can be minimized in $O(n^2)$ time.

Proof: Supposet that T has r rows and t columns. Consider the algorithm of Figure 4. Lines (1)-(3) have a total cost of no more than O(n). In each iteration of the outer loop, the cost of lines (4) and (5) is no more than O(r). The outer loop is executed at most r times to give a total cost of no more than $O(r^2)$. The cost of line (6) is assigned to the rows of T as follows. Whenever a row w of W(b) is compared with some other row of S, the cost of this comparison (i.e., O(t)) is assigned to w. Row w is compared with at most r rows, and the total cost assigned to w is no more than O(rt). Since each row of T appears at most in one of the sets W(b), the total cost of line (6) is $O(r^2t)$. Both rt and r are smaller than n and, therefore, the algorithm has a running time of $O(n^2)$. Since the output of this algorithm is a regular tableau, T can be minimized in time $O(n^2)$ []

6.3 Minimizing Tableaux of Class 2

Let T be a tableau such that every row of T is covered by at most one row besides itself. For each row w of T, let c(w) be the other row that covers w (if no other row of T covers w, then c(w)=w). Note that the function c can be computed in $O(n^2)$ time. Our goal is to find whether a repeated nondistinguished variable b_0 is essential.

Let $W(b_0)$ the set of all the rows containing b_0. Suppose that b_0 is not essential, and let ψ be a folding of T that eliminates b_0. It follows that for all rows w in $W(b_0)$, w is mapped to c(w). We can start to compute the homomorphism h corresponding to ψ as follows.

(1) If a nondistinguished variable b appears in some row w of $W(b_0)$, then h(b) is the symbol appearing in column A of c(w).

(2) If b appears in some row c(w), where w is a row of $W(b_0)$, then $h(b) = b$.

The first rule follows from the fact that w is mapped to c(w). The second rule is valid, since c(w) is in the image of a foldoing and, hence, c(w) must be mapped to itself.

Suppose that by applying these rules to all the variables appearing in w and c(w), for all w in $W(b_0)$, a contradiction is derived. Then a folding that eliminates b_0 does not exist. Similarly, if the rules imply that $h(b_0) = b_0$, then b_0 is essential.

If the rules do not imply that b_0 is essential, then at least we can use them to find some other rows that do not belong to the core of T (i.e., the image of ψ). That is, if h has already been computed for a nondistinguished variable b, and $h(b) \neq b$, then every row w that contains b is mapped to c(w). Thus, let M be a set that initially is equal to $W(b_0)$. As we discover more rows that are not in the core of T, these rows are added to M. Rules (1) and (2) for computing h can be applied to all the rows of M, instead of only to the rows of $W(b_0)$. If a contradiction is derived during this process (or $h(b_0) = b_0$), then ψ cannot be a folding and, hence, b_0 is essential. If no contradiction is derived, let $\overline{M}$ be the value of M when no more rows can be added to M, and neither Rule (1) nor Rule (2) can be applied to rows of M. Define a mapping θ as follows:

$$\theta(w) = c(w) \qquad \text{if } w \in \overline{M}$$

$$\theta(w) = w \qquad \text{if } w \notin \overline{M}$$

Lemma 9: The mapping θ is a containment mapping, and no row of $\overline{M}$ is in the image of θ (provided that $h(b_0) \neq b_0$).

Proof: For all rows w of T, c(w) covers w and, hence, θ satisfies the first two conditions of a containment mapping. Suppose that rows w and x of T have the same nondistinguished variable b in some column A. If both w and x belong to $\overline{M}$, then $\theta(w)$ and $\theta(x)$ have the same symbol in column A (because the rules for computing h do not imply a contradiction). If neither w nor x is in $\overline{M}$, then obviously $\theta(w)$ and $\theta(x)$ have the same symbol in column A. Suppose that w is in $\overline{M}$ but x is not. Therefore, $h(b) = b$ (otherwise all the rows containing b should be in $\overline{M}$) and, hence, $\theta(w)$ and $\theta(x)$ have b in column A. If x is in $\overline{M}$ but w is not, then we get a similar result. Thus, θ satisfies also the third condition of a containment mapping.

Suppose that a row w of $\overline{M}$ is mapped to some other row u of $\overline{M}$. Since u is in $\overline{M}$, it must have some repeated nondistinguished variable b such that $h(b) \neq b$. But since w is mapped to u, Rule (2) implies that $h(b) = b$. However, it is assumed that no contradiction has occured and, hence, w cannot be mapped to u. []

It follows that if the rules do not imply a contradiction and $h(b_0) \neq b_0$, then all the rows of $\overline{M}$ (and hence all the rows containing b_0) can be eliminated from T. If a contradiction is derived, then b_0 is essential. Thus, by repeating this process for every nondistinguished variable in T, we obtain a regular tableau $\overline{T}$ equivalent to T.

The procedure FOLD(b,T) decides whether a nondistinguished variable b is essential. If b is essential, then FOLD(b,T) returns the empty set. If b is not essential, then FOLD(b,T) returns a set of rows (containing all the rows having b) that can be eliminated from T. The complete algorithm is given in Figure 5. It is assumed that the variables of T are represented by the numbers 1,2,...,m. The values of the homomorphism computed by the procedure FOLD(b,T) are stored in the array h; initially all the entries in this array are zero.

<u>Lemma 10</u>: A call FOLD(b,T) requires O(n) time, where n is the size of T.

<u>Proof</u>: Suppose that T has r rows and t columns. For each row w, the cost O(t) of executing the loop of lines (8)-(10) and the loop of lines (11)-(16) is assigned to w. Line (7) is executed at most once for each row and, therefore, the total cost of lines (7)-(16) is O(rt). The cost of finding all the rows that contain d in line (6) is assigned to d. Let cost(d) be that cost. For each symbol we use a linked list that points to all the rows containing that symbol, and since each d is put on QUEUE at most once, $\sum_{d} \text{cost}(d) = O(n)$. Thus, the total cost of the

```
     procedure FOLD(b,T):
     begin
         array h[1:m];
         /* Initially QUEUE is empty */
 (1)     M := $\phi$;
 (2)     h := 0;
 (3)     add b to QUEUE;
 (4)     while QUEUE is not empty do
             begin
 (5)             find and delete d, the first element of QUEUE;
 (6)             for each row w that contains d, and w $\notin$ M do
                     begin
 (7)                     add w to M;
 (8)                     for each repeated nondistinguished
                             variable e that appears in c(w) do
 (9)                         if h(e) = 0 then h(e) := e
(10)                             else if h(e) $\neq$ e then return $\phi$;
(11)                     for each column A of w that has a repeated
                             nondistinguished variable e do
                             begin
(12)                             let s be the symbol of c(w) in column A;
(13)                             if h(e) = 0 then h(e) := s;
(14)                             if h(e) $\neq$ s then return $\phi$;
(15)                             if h(e) $\neq$ e,
                                     and e has not been on QUEUE
(16)                                 then add e to QUEUE;
                             end;
                     end;
             end;
(17)     if h(b) = b then return $\phi$ else return M;
     end FOLD;
     begin /* main procedure */
         /* Initially all the variables are marked "unconsidered". */
(18)     while there is a repeated nondistinguished
               variable b marked "unconsidered" do
             begin
(19)             mark b "considered";
(20)             M := FOLD(b,T);
(21)             delete all the rows of M from T;
             end;
(22)     return T;
     end
```

Figure 5

outer loop (lines (4)-(16)) is O(n). Finally, note that lines (1)-(3) and line (17) have a total cost of no more than O(n). []

Theorem 11: A tableau T, in which each row is covered by at most one row besides itself, can be minimized in time $O(n^2)$.

Proof: Suppose that T′ is the result of applying the algorithm of Figure 5 to T. If a variable b is not essential in T′, then it is not essential in T or in any tableau obtained from T by deleting some redundant rows (because T and T′ are equivalent). Thus, all the rows containing b are deleted in line (21), and hence T′ is regular. By Lemma 10, the running time of this algorithm is $O(n^2)$. []

6.4 Simple Tableaux

Suppose that T is a simple tableau. Let S be a set of rows, and let w be a row of T. The closure of S with respect to w, denoted $CL_w(S)$, is the minimal set of rows such that

(1) $S \subseteq CL_w(S)$, and

(2) if x is a row in $CL_w(S)$ such that x has a repeated nondistinguished variable b in some column A, and w has some other symbol in this column, then all the rows containing b are in $CL_w(S)$.

In [3] it is shown that if w covers $CL_w(S)$, then the tableau obtained by deleting all the rows of $CL_w(S) - w$ is equivalent to T (this is true even if T is not simple). Furthermore, if some repeated nondistinguished variable b of T is not essentail, then there is a row w (that does not contain b) such that w covers $CL_w(W(b))$. (Note that w is not in $CL_w(W(b))$, since w is not in W(b).)

These results can be used to obtain a regular tableau equivalent to T as follows. Compute $CL_w(W(b))$ for each repeated nondistinguished variable b and each row w that does not contain b. If w covers $CL_w(W(b))$, then delete all the rows of $CL_w(W(b))$ from T. The resulting tableau is equivalent to T, and it is regular because all repeated nondistinguished variables that are not essential have been eliminated [3]. In this section we describe an implementation of this algorithm that runs in $O(n^2)$ time.

<u>Lemma 12</u>: Let w be a row of a simple tableau T. Suppose that b_1 and b_2 are two repeated nodistinguished variables of T that do not occur in w. Then $CL_w(W(b_1))$ and $CL_w(W(b_2))$ are either equal or disjoint.

<u>Proof</u>: Let $S_1 = CL_w(W(b_1))$ and $S_2 = CL_w(W(b_2))$. Suppose that some row x belongs to both S_1 and S_2. Row x must have some repeated nondistinguished variable b that does not occur in w and, hence, W(b) is contained in both S_1 and S_2. We claim that both S_1 and S_2 are equal to $CL_w(W(b))$. By the definition of a closure, if R is a subset of $CL_w(S)$ (for any row w and set of rows S), then $CL_w(R) \subseteq CL_w(S)$. Thus, both S_1 and S_2 contain $CL_w(W(b))$.

Let $S = CL_w(W(b))$. Suppose that S_1 is not equal to S. If some row of $W(b_1)$ is in S, then all the rows of $W(b_1)$ are in S, and S is equal to S_1 (because it satisfies the definition of $CL_w(W(b_1))$). Thus $W(b_1)$ must be contained in $S_1 - S$. We will derive a contradiction by showing that $S_1 - S$ satisfies also the second condition of $CL_w(W(b_1))$. Let u be any row of $S_1 - S$. Since u is in S_1, u has a repeated nondistinguished variable $\bar{b}$ in some column A, and w has some other symbol in that column. If some row of $W(\bar{b})$ is in S, then all the rows of $W(\bar{b})$ must be in S, and u cannot be in $S_1 - S$. Thus, $W(\bar{b})$ is disjoint from S and, hence, all the rows containing $\bar{b}$ are in $S_1 - S$ (since they are in S_1). Therefore, $S_1 - S$ satisfies also the second condition of $CL_w(W(b_1))$. Since this is impossible, it follows that S_1 is equal to S. Similarly, S_2 is equal to S. []

Lemma 12 implies that if $CL_w(W(b))$ has been computed, and $\bar{b}$ is a repeated nondistinguished variable that appears in some row of $CL_w(W(b))$, then there is no need to compute $CL_w(W(\bar{b}))$ (it is assumed that neither b nor $\bar{b}$ occur in w). Thus, for each row w we do the following. At first all repeated nondistinguished variables that do not appear in w are marked "unconsidered". The next step is to compute $CL_w(W(b))$ for some repeated nondistinguished variable that is marked "unconsidered". During this step all repeated nondistinguished variabes that occur in some row of $CL_w(W(b))$ are marked "considered". If $CL_w(W(b))$ is covered by w, then all the rows of $CL_w(W(b))$ are deleted. This step is repeated for some other variable that is marked "unconsidered", until all the variables are marked "considered". The complete algorithm is described in Figure 6.

<u>Theorem 13</u>: A simple tableau T can be minimized in $O(n^2)$ time.

<u>Proof</u>: By Lemma 12 and the results of [3], the algorithm of Figure 6 returns a regular tableau equivalent to T. Consider the time complexity of this algorithm. We assume that T has r rows and t columns, and n

```
    procedure CLOSURE(b,w):
    begin
 (1)    S := $\phi$;
 (2)    make QUEUE empty;
 (3)    mark b "considered";
 (4)    add all the rows containing b to QUEUE;
 (5)    while QUEUE is not empty do
            begin
 (6)            let v be the first row on QUEUE;
 (7)            move v from QUEUE to S;
 (8)            for every column A do
 (9)                if v and w disagree in column A, v has a repeated
                       nondistinguished variable d in this column,
                       and d is marked "unconsidered" then
                     begin
(10)                     mark d "considered";
(11)                     for every row x containing d do
(12)                         if x is neither in S nor on QUEUE
(13)                            then add x to QUEUE;
                     end;
            end;
(14)    return S;
    end
    begin /* main procedure */
(15)    for every row w do
            begin
(16)            mark all repeated nondistinguished variables "unconsidered";
(17)            for every repeated nondistinguished variable d
                    that occurs in w do
(18)                mark d "considered";
(19)            while there is a repeated nondistinguished variable b
                      marked "unconsidered" do
                    begin
(20)                    R := CLOSURE(b,w);
(21)                    if R $<$ w then delete all the rows of R from T;
                    end;
            end;
(22)    return T;
    end
```

Figure 6

is the size of T (i.e., $n = O(rt)$). At first we will show that a call CLOSURE(b,w) requires $O(st)$ time, where s is the number of rows in $CL_w(W(b))$. Consider the cost of executing the loop of lines (11)-(13).

We assume that for every variable in T there is a linked list that points to all the rows containing this variable. These lists can be created in O(n) time prior to the execution of the algorithm. By using these lists, the cost of finding all the rows containing d in the loop of lines (11)-(13) is proportional to the number of these rows. The test of deciding whether a row is in S or on QUEUE can be implemented in constant time. Thus, the cost of lines (11)-(13) is accounted for by assigning a constant cost to each row w whenever w is examined in this loop. A row is examined in this loop no more than t times, since it has at most t variables. Also note that if a row is examined in this loop, then it belongs to $CL_w(W(b))$. Therefore, the total cost of lines (11)-(13) is O(st).

Consider now the cost of executing the loop of lines (5)-(13) once (excluding the cost of lines (11)-(13)). Lines (6) and (7) have a constant cost. The cost of lines (8)-(10) is O(t). Since the loop of lines (5)-(13) is repeated s times, the total cost of this loop is O(st). Lines (1)-(4) have a cost of O(s). Thus, a call CLOSURE(b,w) requires O(st) time.

We now compute the cost of executing the loop of lines (15)-(21) once. The cost of line (16) is O(t) (since each column of a simple tableau has at most one repeated nondistinguished variable). The loop of lines (17)-(18) requires O(t) time. The cost of line (19) is no more than the number of repeated nondistinguished variables, i.e., O(t). The cost of executing lines (20)-(21) once is O(st), where s is the number of rows in R. For each row w, all the sets obtained as a value of R in line (20) are pairwise disjoint. Thus, the cost of executing the loop of lines (15)-(21) once is O(rt). This loop is repeated r times and, hence, the total cost is $O(r^2t)$, i.e., no more than $O(n^2)$. []

<u>Theorem 14</u>: If T_1 and T_2 are simple tableaux, then testing whether T_1 is equivalent to T_2 can be done in $O(n^2)$ time.

<u>Proof</u>: By using the algorithm of Figure 6, we compute regular tableaux $\overline{T}_1$ and $\overline{T}_2$ equivalent to T_1 and T_2, respectively. It follows from the results of [3] that testing whether $\overline{T}_1$ is equivalent to $\overline{T}_2$ can be done in $O(n^2)$ time, where n is the size of T_1 and T_2. []

7. Decomposition of Tableaux

Let T be a tableau that does not necessarily belong to one of the classes we have discussed so far. None of the three minimization algorithms can minimize T in polynomial time, but they can be used as heuristics. The minimization algorithm for simple tableaux can be applied to any tableau, and the result is an equivalent tableau possibly with fewer rows. The idea behind the algorithm of Section 6.2 can be used to minimize tableaux as follows. Let b be a repeated nondistinguished variable of a tableau T, and suppose that the set S of all the rows containing b does not have any other repeated nondistinguished variable. Then all the rows of S can be deleted if they are covered by a set of rows that have the same symbol in the column of b.

The algorithm of Section 6.3 is not only good as a heuristic, but can also be used as an exponential time minimization algorithm for tableaux. Let T be a tableau. For every row i of T, we define C(i) to be the set of all the rows that cover row i, i.e.,

$$C(i) = \{j \mid j \neq i \text{ and row } j \text{ covers row } i\}$$

Suppose we construct a function c such that for all i, $c(i) \in C(i)$ (it is understood that $c(i) = i$ if $C(i) = \phi$). Using c we can apply the algorithm of Section 6.3 as a heuristic. The number of all possible c′s is exponential only in the number of rows i for which C(i) has more than one element. If we execute the algorithm once for each possible c we are guaranteed to minimize T, since at least one of these c′s corresponds to a folding from T to its core.

The following approach can be used to further reduce the exponential factor in the running time of this algorithm. We define a relation R on the rows of a tableau T as follows. For rows x and y of T, xRw if and only if x and w have the same nondistinguished variable in some column. Obviously, R is symmetric and reflexive. Let $P_1, P_2, \ldots, P_q$ be the equivalence classes of the transitive closure of R.

<u>Lemma 15</u>: Let θ be a containment mapping from a tableau T to itself, and let k ($1<k<q$) be a fixed integer. Define

$$\xi(x) = \theta(x) \quad \text{if } x \in P_k$$
$$\xi(x) = x \quad \text{otherwise}$$

Then ξ is a containment mapping.

Proof: Obviously, for all rows x, $\xi(x)$ covers x. Suppose that rows x and w have the same nondistinguished variable in column A. Thus, either both w and x are in P_k or both are not in P_k. In either case, $\xi(x)$ and $\xi(w)$ have the same symbol in column A. []

Lemma 15 implies that when the algorithm of Section 6.3 is applied as a heuristic, we have to consider only c's such that for some fixed k, $c(i) \in C(i)$ if row i is in P_k; otherwise $c(i) = i$. If among all the P_j's, P_{j_0} has a maximum number, say m, of rows for which C(i) contains more than one element, then the number of all possible c's is exponential only in m. For each P_j there is at least one possible c that maps all redundant rows in P_j to the core of T. Thus, no c has to be considered more than once.

8. Synthesis of Expressions from Tableaux

Optimal expressions can be synthesized from simple tableaux in $O(n^2)$ time [5]. Suppose we are given a tableau T. The optimization process is done in two steps.

(a) Minimize the number of rows in T.

(b) Produce from T an expression in which select and project are applied as early as possible.

The approach taken in the second step was found useful by previous workers (e.g., [12,17]) in the field of expression optimization, and can be viewed as our "cost function."

In order to obtain an optimal expression for a minimal tableau T, [5] uses the following approach. At first all constants are deleted from the summary of T. Then an optimal expression E is synthesized. Finally a new operator <u>augment</u>, defined by $\alpha_{A=c}(r) = \{\mu \mid \mu(A) = c$ and there exists ν in r such that for all attributes B in the relation scheme of r, $\mu(B) = \nu(B)\}$, is applied to E to introduce the constants that were deleted from the summary of T.

Suppose we decompose a tableau T into equivalence classes $P_1, P_2, \ldots, P_q$ as described in Section 7, and let s be the summary of T. For every j, we can define a tableau T_j that have the same rows as P_j and a summary as follows. For each column A, if s has a nonblank symbol in column A and that symbol appears also in column A of P_j, then the summary of T_j has the same symbol in column A; otherwise, it has a blank. It follows that $T = \bowtie_{j=1}^{q} T_j$. Thus, T comes from an expression if and only if each T_j comes from an expression. Suppose that each T_j comes from the expression E_j. Then T corresponds to the expression $E = \bowtie_{j=1}^{q} E_j$. If T is a minimal tableau, we can get an optimal expression for T as follows. At first all constants are deleted from the summary of T. Then we decompose T and find an optimal expression E_j for each T_j. (Note that each T_j is minimal.) Let $E = \bowtie_{j=1}^{q} E_j$. By applying augmentation to E we get an optimal expression for T.

If T is a tableau with at most one repeated nondistinguished variable in each row, then each T_j is a simple tableau (because it has at most one repeated nondistinguished variable). Thus, we can synthesize an optimal expression for T in polynomial time. However, if T is a tableau in which every row is covered by at most one row besides itself, then testing whether T comes from an expression is NP-complete.(1)

Theorem 13: Let T be a tableau in which every row is covered by at most one row besides itself. The problem whether T corresponds to a restricted relational expression is NP-complete.

Proof: The problem of testing whether a tableau T corresponds to an expression (shown NP-complete in [19]) is reduced to this problem as follows. Let T be a tableau, and let G be a new attribute. We construct a tableau T′ by adding to T a new column that corresponds to G. The summary of T′ has a blank, and row i has the constant i in this column. We claim that T corresponds to an expression if and only if T′ corresponds to an expression.

If. Let E′ be an expression for T′. Delete G from each relation scheme in E′, and delete every selection operator that is applied to the attribute G. Each projection operator π_X that appears in E′ is replaced with $\pi_{X-\{G\}}$. Let E be the resulting expression. It is easy to show that E corresponds to T.

Only if. Let E be an expression for T. Let R_i be the relation scheme that corresponds to row i of T, and define $R_i' = R_i \cup \{G\}$ for every i. An expression E′ for T′ is obtained from E by replacing each R_i with $\pi_{R_i}(\sigma_{G=i}(R_i'))$.

(1) See [2,11] for an exposition of NP-completeness and related topics.

Since each row of T′ is covered by no row besides itself, and an expression for T can be obtained from an expression for T′ in polynomial time, the proof is complete. []

References

[1] A. V. Aho, C. Beeri, and J. D. Ullman, "The Theory of Joins in Relational Databases," ACM Trans. on Database Systems, Vol. 4, No. 3 (1979), pp. 297-314.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974.

[3] A. V. Aho, Y. Sagiv, and J. D. Ullman, "Equivalences Among Relational Expressions," SIAM J. Computing, Vol. 8, No. 2 (1979), pp. 218-246.

[4] A. V. Aho, Y. Sagiv, and J. D. Ullman, "Efficient Optimization of a Class of Relational Expressions," to appear in ACM Trans. on Database Systems.

[5] A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman, "Inferring a Tree from Lowest Common Ancestors with an Application to the Optimization of Relational Expressions," Proc. Sixteenth Annual Allerton Conf. on Communication, Control and Computing, October, 1978, pp. 54-63.

[6] W. W. Armstrong, "Dependency Structures of Data Base Relationship," Proc. IFIP 74, North Holland, 1974, pp. 580-583.

[7] C. Beeri, R. Fagin, and J. H. Howard, "A Complete Axiomatization

for Functional and Multivalued Dependencies," Proc. ACM-SIGMOD International Conference on the Management of Data, August, 1977, pp. 47-61.

[8] A. K. Chandra and P. M. Merlin, "Optimal Implementation of Conjunctive Queries in Relational Data Bases," Proc. Ninth Annual ACM Symposium on Theory of Computing, May, 1977, pp. 77-90.

[9] E. F. Codd, "A Relational Model for Large Shared Data Banks," Comm. ACM Vol. 13, No. 6 (1970), pp. 377-387.

[10] E. F. Codd, "Relational Completeness of Data Base Sublanguages," in Data Base Systems (R. Rustin, ed.), Prentice Hall, Englewood Cliffs, NJ, 1972, pp. 65-98.

[11] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1978.

[12] P. A. V. Hall, "Optimization of a Single Relational Expression in a Relational Database System," IBM J. Research and Development, Vol. 20, No. 3 (1976), pp. 244-257.

[13] J. Minker, "Performing Inferences Over Relational Databases," Proc. ACM-SIGMOD International Conference on Management of Data May, 1975, pp. 79-91.

[14] F. P. Palermo, "A Database Search Problem," in Information Systems COINS IV, (J. T. Tou, ed.), Plenum Press, New York, 1974.

[15] R. M. Pecherer, "Efficient Evaluation of Expressions in a Relational Algebra," Proc. ACM Pacific Conference, April, 1975, pp. 44-49.

[16] Y. Sagiv and M. Yannakakis, "Equivalences Among Relational

Expressions With the Union and Difference Operators," to appear in _JACM_.

[17] J. M. Smith and P. Y.-T. Chang, "Optimizing the Performance of a Relational Algebra Database Interface," _Comm. ACM_, Vol. 18, No. 10 (1975), pp. 568-579.

[18] E. Wong and K. Youssefi, "Decomposition - A Strategy for Query Processing, _ACM Trans. on Database Systems_, Vol. 1, No. 3 (1976), pp. 223-241.

[19] M. Yannakakis, private communication.

[20] M. M. Zloof, "Query-by-Example: the Invocation and Definition of Tables and Forms, _Proc. ACM International Conf. on Very Large Data Bases_, Sept., 1975, pp. 1-24.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-79-992	2.	3. Recipient's Accession No.

I. Title and Subtitle
Quadratic Algorithms for Minimizing Joins in Restricted Relational Expressions

5. Report Date
October 1979

6.

'. Author(s)
Yehoshua Sagiv

8. Performing Organization Rept. No.

'. Performing Organization Name and Address

10. Project/Task/Work Unit No.

11. Contract/Grant No.

12. Sponsoring Organization Name and Address
Department of Computer Science
University of Illinois
at Urbana-Champaign
Urbana, IL 61801

13. Type of Report & Period Covered

14.

15. Supplementary Notes

16. Abstracts

An important step in the optimization of queries in relational databases is minimizing the number of join operations used in the evaluation of a query. It is shown that three subclasses of tableaux (including the subclass of simple tableaux) have $O(n^2)$ time equivalence and minimization algorithms. Since tableaux are nonprocedural representations of relational expressions over select, project and join, these minimization algorithms can be used to minimize the number of join operators in expressions whose tableaux belong to one of these subclasses.

7. Key Words and Document Analysis. 17a. Descriptors

relational database, relational algebra, query optimization,

equivalence of queries, conjunctive query, tableau, NP-complete.

7b. Identifiers/Open-Ended Terms

7c. COSATI Field/Group

8. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 42
	20. Security Class (This Page UNCLASSIFIED	22. Price